第10章　容器构件GtkContainer

10.1　事件盒构件GtkEventBox

一些GTK构件没有与之相关联的X窗口，所以它们只在其父构件上显示其外观。由于这个原因，它们不能接收任何事件，并且，如果它们尺寸设置不正确，它们也不会自动剪裁，这样可能会把界面弄得乱糟糟的。如果要想构件接收事件，可以使用事件盒构件(GtkEventBox)。

初一看，GtkEventBox构件好像完全没有什么用。它在屏幕上什么也不画，并且对事件也不做响应。但是，它有一个功能：为它的子构件提供一个 X窗口。因为许多GTK构件并没有相关联的X窗口，所以这一点很重要。虽然没有 X窗口会节省内存，提高系统性能，但它也有一些弱点。没有X窗口的构件不能接收事件，并且对它的任何内容不实施剪裁。事件盒构件的名称强调它的事件处理功能，同时，它也能用于剪裁构件。

用以下函数创建一个新的事件盒构件：

```
GtkWidget *gtk_event_box_new( void );
```

然后子构件就可以添加到GtkEventBox里面：

```
gtk_container_add( GTK_CONTAINER(event_box), child_widget );
```

下面的示例演示了事件盒的用途：创建一个标签，将它剪裁，放到一个小盒子里面，然后设置让鼠标点击时程序退出。改变窗口的尺寸会使标签构件的尺寸发生变化。

```
/* 事件盒构件示例开始eventbox.c */
#include <gtk/gtk.h>
int
main (int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *event_box;
    GtkWidget *label;
    gtk_init (&argc, &argv);
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Event Box");

    gtk_signal_connect (GTK_OBJECT (window), "destroy",
                        GTK_SIGNAL_FUNC (gtk_exit), NULL);

    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    /* 创建一个事件盒，然后将它加到顶级窗口上 */
    event_box = gtk_event_box_new ();
    gtk_container_add (GTK_CONTAINER(window), event_box);
    gtk_widget_show (event_box);

    /* 创建一个长标签 */
```

```
    label = gtk_label_new ("Click here to quit, quit, quit, quit, quit");
    gtk_container_add (GTK_CONTAINER (event_box), label);
    gtk_widget_show (label);

    /* 将标签剪裁短 */
    gtk_widget_set_usize (label, 110, 20);

    /* 为它绑定一个动作*/
    gtk_widget_set_events (event_box, GDK_BUTTON_PRESS_MASK);
    gtk_signal_connect (GTK_OBJECT(event_box), "button_press_event",
                        GTK_SIGNAL_FUNC (gtk_exit), NULL);

    /* 为事件盒创建一个X窗口 */
    gtk_widget_realize (event_box);
    gdk_window_set_cursor (event_box->window, gdk_cursor_new (GDK_HAND1));
    gtk_widget_show (window);
    gtk_main ();
    return(0);
}
/* 示例结束 */
```

将上面的代码保存为eventbox.c，然后写一个下面这样的Makefile：

```
CC = gcc
eventbox: eventbox.c
   $(CC) `gtk-config --cflags`  eventbox.c -o eventbox \
         `gtk-config --libs`
clean:
   rm -f *.o eventbox
```

编译之后，运行结果如图10-1所示。改变窗口大小时，标签所显示的文本数量也会改变。点击标签，应用程序会退出。

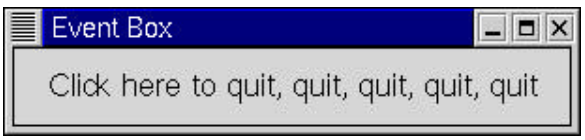

图10-1 使用事件盒，可以让标签构件对点击进行响应

10.2 对齐构件GtkAlignment

GtkAlignment(对齐构件)允许将一个构件放在相对于对齐构件窗口的某个位置和尺寸上。例如，将一个构件放在窗口的正中间时，就要使用对齐构件。

只有如下两个函数与对齐构件相关联：

```
GtkWidget* gtk_alignment_new( gfloat xalign,
                              gfloat yalign,
                              gfloat xscale,
                              gfloat yscale );

void gtk_alignment_set( GtkAlignment *alignment,
```

```
                    gfloat        xalign,
                    gfloat        yalign,
                    gfloat        xscale,
                    gfloat        yscale );
```

第一个函数用指定的参数创建新的对齐构件。第二个函数用于改变对齐构件的参数。

上面函数的所有四个参数都是介于0.0与1.0间的浮点数。xalign 和 yalign参数影响对齐构件内部的构件位置。xscale和yscale 参数影响分配给构件的空间总数。

使用以下函数可以将子构件添加到对齐构件中：

```
gtk_container_add( GTK_CONTAINER(alignment), child_widget );
```

要看关于对齐构件的例子，请看进度条构件的示例。

10.3 框架构件GtkFrame

GtkFrame(框架构件)可以用于在盒子中封装一个或一组构件，框架本身还可以有一个标签。标签的位置和风格可以灵活改变。

用下面函数可以创建新框架构件：

```
GtkWidget *gtk_frame_new( const gchar *label );
```

其中，label参数是框架构件的标签。

缺省设置时，标签放在框架的左上角。传递 NULL时，框架不显示标签。标签文本可以用下面的函数改变。

```
void gtk_frame_set_label( GtkFrame    *frame,
                          const gchar *label );
```

标签的位置可以用下面的函数改变：

```
void gtk_frame_set_label_align( GtkFrame *frame,
                                gfloat    xalign,
                                gfloat    yalign );
```

xalign和yalign参数取值范围介于0.0和1.0之间。xalign指定标签在框架构件上部水平线上的位置。yalign参数目前还没有使用。xalign的缺省值是0.0，它将标签放在框架构件的左上角处。

下面的函数用于改变frame的轮廓框的风格。

```
void gtk_frame_set_shadow_type( GtkFrame      *frame,
                                GtkShadowType  type);
```

Type参数可以取以下值：

GTK_SHADOW_NONE

GTK_SHADOW_IN

GTK_SHADOW_OUT

GTK_SHADOW_ETCHED_IN (缺省值)

GTK_SHADOW_ETCHED_OUT

下面的例子演示了怎样使用框架构件。

```
/* 框架构件示例开始 frame.c */
#include <gtk/gtk.h>

int main( int   argc,
          char *argv[] )
```

```
{

  /* 构件的存储类型是GtkWidget */
  GtkWidget *window;
  GtkWidget *frame;
  GtkWidget *button;
  gint i;

  /* 初始化GTK */
  gtk_init(&argc, &argv);
  /* 创建一个新窗口 */
  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(window), "Frame Example");

  /* 为窗口的"destory"事件设置一个回调函数*/
  gtk_signal_connect (GTK_OBJECT (window), "destroy",
                      GTK_SIGNAL_FUNC (gtk_main_quit), NULL);

  gtk_widget_set_usize(window, 300, 300);
  /* 设置窗口的边框宽度*/
  gtk_container_set_border_width (GTK_CONTAINER (window), 10);

  /* 创建一个框架构件 */
  frame = gtk_frame_new(NULL);
  gtk_container_add(GTK_CONTAINER(window), frame);

  /* 设置框架的标签*/
  gtk_frame_set_label( GTK_FRAME(frame), "GTK Frame Widget" );

  /* 将标签在框架构件的右边对齐 */
  gtk_frame_set_label_align( GTK_FRAME(frame), 1.0, 0.0);

  /* 设置框架构件的风格*/
  gtk_frame_set_shadow_type( GTK_FRAME(frame), GTK_SHADOW_ETCHED_OUT);
  gtk_widget_show(frame);

  /* 显示窗口 */
  gtk_widget_show (window);

  /* 进入主循环*/
  gtk_main ();

  return(0);
}
/* 示例结束 */
```

将上述代码保存为frame.c，然后编写一个像下面这样的Makefile文件。

```
CC = gcc
frame: frame.c
   $(CC) `gtk-config --cflags`  frame.c -o frame \
```

图10-2　框架构件

```
K_C

/*      一个比例框架构件，将它
asp     C_      -  gtk_
```

```
                                        0.5, /* 水向居中 */
                                        0.5, /* 竖向居中 */
                                        2, /* xsize/ysize = 2 */
                                        FALSE /*忽略子构件的比例*/);

    gtk_container_add (GTK_CONTAINER(window), aspect_frame);
    gtk_widget_show (aspect_frame);

    /* 添加一个子构件到比例框架构件中*/
    drawing_area = gtk_drawing_area_new ();

    /* 要求画一个200×200的窗口，但是比例框架构件会给出一个200×100
     * 的窗口，因为我们已经指定了2×1的比例值 */
    gtk_widget_set_usize (drawing_area, 200, 200);
    gtk_container_add (GTK_CONTAINER(aspect_frame), drawing_area);
    gtk_widget_show (drawing_area);

    gtk_widget_show (window);
    gtk_main ();
    return 0;
}
/* 示例结束 */
```

将上面的代码保存为aspectframe.c，然后编写一个如下所示的Makefile文件。

```
CC = gcc
aspectframe: aspectframe.c
   $(CC) `gtk-config --cflags`  aspectframe.c \
-o aspectframe `gtk-config --libs`

clean:
   rm -f *.o aspectframe
```

编译后，执行结果见图10-3。试着调整窗口的尺寸，看一看框架构件如何变化。

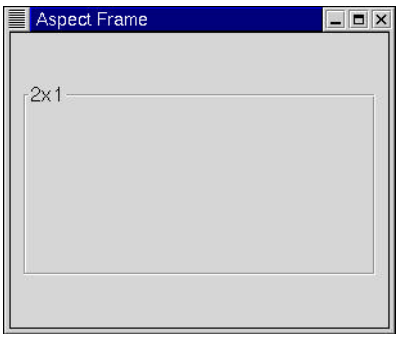

图10-3 比例框架构件

10.5 分栏窗口构件GtkPanedWindow

如果想要将一个窗口分成两个部分，可以使用 GtkPanedWindow(分栏窗口构件)。窗口两部分的尺寸由用户控制，它们之间有一个凹槽，上面有一个手柄，用户可以拖动此手柄改变

两部分的比例。窗口划分可以是水平 (HPaned)或垂直的(VPaned)。有两种分栏窗口构件：GtkHPaned（水平分栏窗口构件）和GtkVPaned（垂直分栏窗口构件）。

用以下函数创建新的分栏窗口构件：

```
GtkWidget *gtk_hpaned_new (void); 水平分栏窗口构件*/
GtkWidget *gtk_vpaned_new (void); 垂直分栏窗口构件*/
```

创建了分栏窗口构件后，可以在它的两边添加子构件。用下面的函数完成：

```
void gtk_paned_add1 (GtkPaned *paned,
                     GtkWidget *child);

void gtk_paned_add2 (GtkPaned *paned,
                     GtkWidget *child);
```

gtk_paned_add1() 将子构件添加到分栏窗口构件的左边或顶部。

gtk_paned_add2() 将子构件添加到分栏窗口构件的右边或下部。

如果希望在分栏窗口构件上设置很复杂的界面，可以将该构件和组装盒（ GtkHBox和GtkVBox）或表格构件结合使用。

分栏窗口构件的视觉外观可以用以下函数改变：

```
void gtk_paned_set_handle_size( GtkPaned *paned,
                                guint16   size);

void gtk_paned_set_gutter_size( GtkPaned *paned,
                                guint16   size);
```

第一个函数设置手柄的尺寸，第二个函数设置两部分之间的凹槽的尺寸。

在下面的例子中，创建了一个假想的用户 Email程序的用户界面。窗口被垂直划分为两个部分，上面部分显示一个 Email信息列表，下部显示 Email文本信息。所有的程序都是相当直接的。有几点要注意：在文本构件显现前文本不能加到文本构件中，但你可以调用gtk_widget_realize()函数完成，不过，作为一个技巧，我们可以为构件的“ realize ”信号设置一个信号处理函数，并在这个函数里面添加文本；还有，我们需要为包含文本窗口和它的滚动条的表格构件（GtkTable）设置GTK_SHRINK选项，以便当窗口的底部变小时，能够正确显示，而不是将下部的构件压到窗口的底部去。

```
/* 分栏窗口构件示例 paned.c */
#include <gtk/gtk.h>

/* 创建一个"信息"列表 */
GtkWidget *
create_list (void)
{

    GtkWidget *scrolled_window;
    GtkWidget *list;
    GtkWidget *list_item;

    int i;
    char buffer[16];
   /* 创建一个新的分栏窗口构件，只有需要时，滚动条才出现 */
    scrolled_window = gtk_scrolled_window_new (NULL, NULL);
```

```
    gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scrolled_window),
                                    GTK_POLICY_AUTOMATIC,
                                    GTK_POLICY_AUTOMATIC);

    /*创建一个新的列表框，将它放在分栏窗口构件中 */
    list = gtk_list_new ();
    gtk_scrolled_window_add_with_viewport (
              GTK_SCROLLED_WINDOW (scrolled_window), list);
    gtk_widget_show (list);

    /*在列表中添加一些列表项*/
    for (i=0; i<10; i++) {
        sprintf(buffer,"Message #%d",i);
        list_item = gtk_list_item_new_with_label (buffer);
        gtk_container_add (GTK_CONTAINER(list), list_item);
        gtk_widget_show (list_item);

    }

    return scrolled_window;
}

/* 向文本构件中添加一些文本时必须先用一个回调函数实现文本构件。
 * 添加文本时，先将构件冻结，添加完成后，将构件解冻 */
 void realize_text (GtkWidget *text, gpointer data)
{
    gtk_text_freeze (GTK_TEXT (text));
    gtk_text_insert (GTK_TEXT (text), NULL, &text->style->black, NULL,
    "From: pathfinder@nasa.gov\n"
    "To: mom@nasa.gov\n"
    "Subject: Made it!\n"
    "\n"
    "We just got in this morning. The weather has been\n"
    "great - clear but cold, and there are lots of fun sights.\n"
    "Sojourner says hi. See you soon.\n"
    " -Path\n", -1);

    gtk_text_thaw (GTK_TEXT (text));
}

/*创建一个滚动文本区域，用于显示添加的信息 */
GtkWidget *
create_text (void)
{
    GtkWidget *table;
    GtkWidget *text;
    GtkWidget *hscrollbar;
    GtkWidget *vscrollbar;

    /* 创建一个table构件，用于容纳文本构件和滚动条 */
    table = gtk_table_new (2, 2, FALSE);
```

```
    /* 将文本构件放在table的左角，注意在
     * y方向设为GTK_SHRINK */
    text = gtk_text_new (NULL, NULL);
    gtk_table_attach (GTK_TABLE (table), text, 0, 1, 0, 1,
                      GTK_FILL | GTK_EXPAND,
                      GTK_FILL | GTK_EXPAND | GTK_SHRINK, 0, 0);
    gtk_widget_show (text);

    /* 将一个水平滚动条放在左下角*/
    hscrollbar = gtk_hscrollbar_new (GTK_TEXT (text)->hadj);
    gtk_table_attach (GTK_TABLE (table), hscrollbar, 0, 1, 1, 2,
                      GTK_EXPAND | GTK_FILL, GTK_FILL, 0, 0);
    gtk_widget_show (hscrollbar);

    /*将一个垂直滚动条放在右上角*/
    vscrollbar = gtk_vscrollbar_new (GTK_TEXT (text)->vadj);
    gtk_table_attach (GTK_TABLE (table), vscrollbar, 1, 2, 0, 1,
                      GTK_FILL, GTK_EXPAND | GTK_FILL | GTK_SHRINK, 0, 0);
    gtk_widget_show (vscrollbar);

    /*将一个处理函数添加到文本构件，当它实现时在文本构件上显示一段信息 */
    gtk_signal_connect (GTK_OBJECT (text), "realize",
                        GTK_SIGNAL_FUNC (realize_text), NULL);
    return table;
}

int main (int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *vpaned;
    GtkWidget *list;
    GtkWidget *text;

    gtk_init (&argc, &argv);
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Paned Windows");
    gtk_signal_connect (GTK_OBJECT (window), "destroy",
                        GTK_SIGNAL_FUNC (gtk_main_quit), NULL);

    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    gtk_widget_set_usize (GTK_WIDGET(window), 450, 400);
    /*在顶级窗口上添加一个垂直分栏窗口构件*/
    vpaned = gtk_vpaned_new ();
    gtk_container_add (GTK_CONTAINER(window), vpaned);
    gtk_paned_set_handle_size (GTK_PANED(vpaned),
                               10);
    gtk_paned_set_gutter_size (GTK_PANED(vpaned),
                               15);
    gtk_widget_show (vpaned);

    /*在分格窗口的两边添加一些构件*/
```

```
    list = create_list ();
    gtk_paned_add1 (GTK_PANED(vpaned), list);
    gtk_widget_show (list);
       text = create_text ();
    gtk_paned_add2 (GTK_PANED(vpaned), text);
    gtk_widget_show (text);
    gtk_widget_show (window);
    gtk_main ();
    return 0;
}
/* 示例结束 */
```

编译后的运行结果如图 10-4所示。尝试改变窗口尺寸，观察上下两部分是怎么变的，再试着改变一下上下两部分的相对比例，看看其效果。

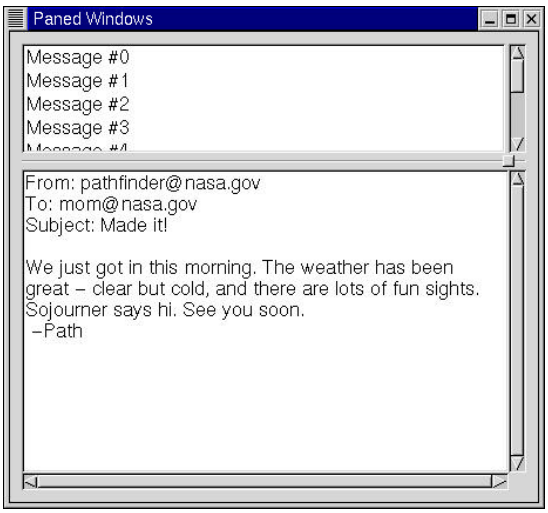

图10-4 分栏窗口示例

10.6 视角构件GtkViewport

一般很少直接使用 GtkViewport(视角构件)。多数情况下是使用分栏窗口构件，由分栏窗口构件使用视角构件。

视角构件允许在其中放置一个较大的构件，这样可以一次只看到构件的一部分。它用调整对象定义要显示的区域。

用以下函数创建视角构件。

```
GtkWidget *gtk_viewport_new( GtkAdjustment *hadjustment,
                             GtkAdjustment *vadjustment );
```

可以看到，创建构件时能够指定构件使用的水平和垂直调整对象。如果给函数传递一个NULL参数，构件会自己创建一个调整对象。

创建构件后，可以用以下函数设置和取得它的调整对象：

```
GtkAdjustment *gtk_viewport_get_hadjustment (GtkViewport *viewport );
GtkAdjustment *gtk_viewport_get_vadjustment (GtkViewport *viewport );
void gtk_viewport_set_hadjustment( GtkViewport    *viewport,
```

```
                                    GtkAdjustment *adjustment );
void gtk_viewport_set_vadjustment( GtkViewport   *viewport,
                                    GtkAdjustment *adjustment );
```

下面的函数可用于改变视角构件的外观：

```
void gtk_viewport_set_shadow_type( GtkViewport   *viewport,
                                    GtkShadowType  type );
```

Type参数可以取以下值：

GTK_SHADOW_NONE,

GTK_SHADOW_IN,

GTK_SHADOW_OUT,

GTK_SHADOW_ETCHED_IN,

GTK_SHADOW_ETCHED_OUT

10.7 滚动窗口构件GtkScrolledWindow

GtkScrolledWindow(滚动窗口构件)用于创建一个可滚动区域，并将其他构件放入其中。可以在滚动窗口中插入任何其他构件，在其内部的构件不论尺寸大小都可以通过滚动条访问到。

用下面的函数创建新的滚动窗口构件。

```
GtkWidget *gtk_scrolled_window_new( GtkAdjustment *hadjustment,
                                    GtkAdjustment *vadjustment );
```

第一个参数是水平方向的调整对象，第二个参数是垂直方向的调整对象。它们总是设置为NULL。

```
void gtk_scrolled_window_set_policy( GtkScrolledWindow *scrolled_window,
                                     GtkPolicyType      hscrollbar_policy,
                                     GtkPolicyType      vscrollbar_policy );
```

这个函数可以设置滚动条出现的方式。第一个参数是要设置的滚动窗口构件，第二个设置水平滚动条出现的方式，第三个参数设置垂直滚动条的方式。

滚动条的方式取值可以为GTK_POLICY_AUTOMATIC或GTK_POLICY_ALWAYS。当要求滚动条根据需要自动出现时，可设为GTK_POLICY_AUTOMATIC；若设为GTK_POLICY_ALWAYS，滚动条会一直出现在滚动窗口构件上。

可以用以下函数将构件放在滚动窗口构件上：

```
void gtk_scrolled_window_add_with_viewport(
GtkScrolledWindow *scrolled_window,
GtkWidget         *child);
```

下面是一个简单例子：在滚动窗口构件中放置一个表格构件，并在表格中放 100个开关按钮。

```
/* 滚动窗口示例开始 scrolledwin.c */
#include <gtk/gtk.h>
void destroy(GtkWidget *widget, gpointer data)
{
    gtk_main_quit();
}
```

```
int main (int argc, char *argv[])
{
    static GtkWidget *window;
    GtkWidget *scrolled_window;
    GtkWidget *table;
    GtkWidget *button;
    char buffer[32];
    int i, j;

    gtk_init (&argc, &argv);

    /* 创建一个新的对话框构件，滚动窗口构件就放在这个窗口上 */
    window = gtk_dialog_new ();
    gtk_signal_connect (GTK_OBJECT (window), "destroy",
                        (GtkSignalFunc) destroy, NULL);
    gtk_window_set_title (GTK_WINDOW (window), "GtkScrolledWindow example");
    gtk_container_set_border_width (GTK_CONTAINER (window), 0);
    gtk_widget_set_usize(window, 300, 300);
    /* 创建一个新的滚动窗口构件 */
    scrolled_window = gtk_scrolled_window_new (NULL, NULL);
    gtk_container_set_border_width (GTK_CONTAINER (scrolled_window), 10);

    /* 滚动条的出现方式可以是GTK_POLICY_AUTOMATIC或GTK_POLICY_ALWAYS。
* 设为GTK_POLICY_AUTOMATIC将自动决定是否需要出现滚动条
* 而设为GTK_POLICY_ALWAYS，将一直显示一个滚动条
* 第一个是水平滚动条，第二个是垂直滚动条
* */
    gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scrolled_window),
                                    GTK_POLICY_AUTOMATIC,
                                    GTK_POLICY_ALWAYS);
    /* 对话框窗口内部包含一个vbox构件*/
    gtk_box_pack_start (GTK_BOX (GTK_DIALOG(window)->vbox), scrolled_window,
                        TRUE, TRUE, 0);
    gtk_widget_show (scrolled_window);

    /* 创建一个包含10×10个方格的表格GtkTable */
    table = gtk_table_new (10, 10, FALSE);

    /* 设置x和y方向的行间间距为10像素 */
    gtk_table_set_row_spacings (GTK_TABLE (table), 10);
    gtk_table_set_col_spacings (GTK_TABLE (table), 10);

    /* 将表格组装到滚动窗口构件中 */
    gtk_scrolled_window_add_with_viewport (
                  GTK_SCROLLED_WINDOW (scrolled_window), table);
    gtk_widget_show (table);

    /* 在表格中添加切换按钮以展示滚动窗口构件 */
    for (i = 0; i < 10; i++)
       for (j = 0; j < 10; j++) {
          sprintf (buffer, "button (%d,%d)\n", i, j);
```

```
            button = gtk_toggle_button_new_with_label (buffer);
            gtk_table_attach_defaults (GTK_TABLE (table), button,
                                       i, i+1, j, j+1);
            gtk_widget_show (button);
         }

      /* 在对话框的底部添加一个"close"按钮 */
      button = gtk_button_new_with_label ("close");
      gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
                                 (GtkSignalFunc) gtk_widget_destroy,
                                 GTK_OBJECT (window));

      /* 让按钮成为缺省按钮 */

      GTK_WIDGET_SET_FLAGS (button, GTK_CAN_DEFAULT);
      gtk_box_pack_start (GTK_BOX (GTK_DIALOG (window)->action_area),
                          button, TRUE, TRUE, 0);

      /* 让按钮固定为缺省按钮，只要按回车键就相当于点击了这个按钮 */
      gtk_widget_grab_default (button);
      gtk_widget_show (button);
      gtk_widget_show (window);
      gtk_main();
      return(0);
   }
   /* 示例结束 */
```

编译后，示例的运行结果如图10-5所示。尝试改变窗口的大小，可以看到滚动条是如何起作用的。还可以用 gtk_widget_set_usize() 函数设置窗口或构件的缺省尺寸。

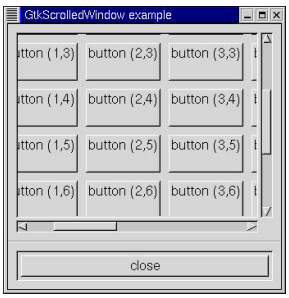

图10-5 滚动窗口构件

10.8 按钮盒构件GtkButtonBox

GtkButtonBox(按钮盒构件)可以很方便地快速布置一组按钮。它有水平和垂直两种样式。你可以用以下函数创建垂直或水平按钮盒：

```
GtkWidget *gtk_hbutton_box_new( void );
GtkWidget *gtk_vbutton_box_new( void );
```

与按钮盒相关的唯一属性是按钮如何放置。可以用以下函数改变按钮间的间距：

```
void gtk_hbutton_box_set_spacing_default( gint spacing );
void gtk_vbutton_box_set_spacing_default( gint spacing );
```

与此相似，可以用下面的函数取得当前的间距值：

```
gint gtk_hbutton_box_get_spacing_default( void );
gint gtk_vbutton_box_get_spacing_default( void );
```

我们能访问的第二个属性会影响按钮在按钮盒中的布局。可以用下面的函数设置它：

```
void gtk_hbutton_box_set_layout_default( GtkButtonBoxStyle layout );
void gtk_vbutton_box_set_layout_default( GtkButtonBoxStyle layout );
```

layout参数可以取以下值：

GTK_BUTTONBOX_DEFAULT_STYLE

GTK_BUTTONBOX_SPREAD
GTK_BUTTONBOX_EDGE
GTK_BUTTONBOX_START
GTK_BUTTONBOX_END
当前的布局设置可以用以下函数取得：

```
GtkButtonBoxStyle gtk_hbutton_box_get_layout_default( void );
GtkButtonBoxStyle gtk_vbutton_box_get_layout_default( void );
```

用以下函数可以将按钮添加到按钮盒中：

```
    gtk_container_add( GTK_CONTAINER(button_box), child_widget );
```

下面的例子演示了按钮盒的不同布局设置。

```
/* 按钮盒示例开始buttonbox.c */
#include <gtk/gtk.h>

/* 用指定的参数创建一个按钮盒 */
GtkWidget *create_bbox (gint  horizontal,
                        char* title,
                        gint  spacing,
                        gint  child_w,
                        gint  child_h,
                        gint  layout)
{
  GtkWidget *frame;
  GtkWidget *bbox;
  GtkWidget *button;

  frame = gtk_frame_new (title);

  if (horizontal)
    bbox = gtk_hbutton_box_new ();
  else
    bbox = gtk_vbutton_box_new ();
  gtk_container_set_border_width (GTK_CONTAINER (bbox), 5);
  gtk_container_add (GTK_CONTAINER (frame), bbox);

  /* 设置按钮盒的外观 */
  gtk_button_box_set_layout (GTK_BUTTON_BOX (bbox), layout);
  gtk_button_box_set_spacing (GTK_BUTTON_BOX (bbox), spacing);
  gtk_button_box_set_child_size (GTK_BUTTON_BOX (bbox), child_w, child_h);

  button = gtk_button_new_with_label ("OK");
  gtk_container_add (GTK_CONTAINER (bbox), button);

  button = gtk_button_new_with_label ("Cancel");
  gtk_container_add (GTK_CONTAINER (bbox), button);

  button = gtk_button_new_with_label ("Help");
  gtk_container_add (GTK_CONTAINER (bbox), button);
  return(frame);
```

```
}

int main( int   argc,
          char *argv[] )
{
  static GtkWidget* window = NULL;
  GtkWidget *main_vbox;
  GtkWidget *vbox;
  GtkWidget *hbox;
  GtkWidget *frame_horz;
  GtkWidget *frame_vert;

  /* 初始化GTK */
  gtk_init( &argc, &argv );
  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title (GTK_WINDOW (window), "Button Boxes");

  gtk_signal_connect (GTK_OBJECT (window), "destroy",
                      GTK_SIGNAL_FUNC(gtk_main_quit),
                      NULL);

  gtk_container_set_border_width (GTK_CONTAINER (window), 10);
  main_vbox = gtk_vbox_new (FALSE, 0);
  gtk_container_add (GTK_CONTAINER (window), main_vbox);

  frame_horz = gtk_frame_new ("Horizontal Button Boxes");
  gtk_box_pack_start (GTK_BOX (main_vbox), frame_horz, TRUE, TRUE, 10);

  vbox = gtk_vbox_new (FALSE, 0);
  gtk_container_set_border_width (GTK_CONTAINER (vbox), 10);
  gtk_container_add (GTK_CONTAINER (frame_horz), vbox);

  gtk_box_pack_start (GTK_BOX (vbox),
          create_bbox (TRUE, "Spread (spacing 40)",
          40, 85, 20, GTK_BUTTONBOX_SPREAD),
          TRUE, TRUE, 0);
  gtk_box_pack_start (GTK_BOX (vbox),
          create_bbox (TRUE, "Edge (spacing 30)",
          30, 85, 20, GTK_BUTTONBOX_EDGE),
          TRUE, TRUE, 5);

  gtk_box_pack_start (GTK_BOX (vbox),
          create_bbox (TRUE, "Start (spacing 20)",
          20, 85, 20, GTK_BUTTONBOX_START),
          TRUE, TRUE, 5);

  gtk_box_pack_start (GTK_BOX (vbox),
          create_bbox (TRUE, "End (spacing 10)",
          10, 85, 20, GTK_BUTTONBOX_END),
          TRUE, TRUE, 5);
```

```
  frame_vert = gtk_frame_new ("Vertical Button Boxes");
  gtk_box_pack_start (GTK_BOX (main_vbox), frame_vert, TRUE, TRUE, 10);
  hbox = gtk_hbox_new (FALSE, 0);
  gtk_container_set_border_width (GTK_CONTAINER (hbox), 10);
  gtk_container_add (GTK_CONTAINER (frame_vert), hbox);

  gtk_box_pack_start (GTK_BOX (hbox),
          create_bbox (FALSE, "Spread (spacing 5)",
          5, 85, 20, GTK_BUTTONBOX_SPREAD),
          TRUE, TRUE, 0);

  gtk_box_pack_start (GTK_BOX (hbox),
          create_bbox (FALSE, "Edge (spacing 30)",
          30, 85, 20, GTK_BUTTONBOX_EDGE),
          TRUE, TRUE, 5);

  gtk_box_pack_start (GTK_BOX (hbox),
          create_bbox (FALSE, "Start (spacing 20)",
          20, 85, 20, GTK_BUTTONBOX_START),
          TRUE, TRUE, 5);

  gtk_box_pack_start (GTK_BOX (hbox),
          create_bbox (FALSE, "End (spacing 20)",
          20, 85, 20, GTK_BUTTONBOX_END),
          TRUE, TRUE, 5);
  gtk_widget_show_all (window);

  /* 进入事件循环 */
  gtk_main ();
  return(0);
}
/* 示例结束 */
```

以上代码的执行效果如图10-6所示。其中演示了按钮盒的各种组织方式。

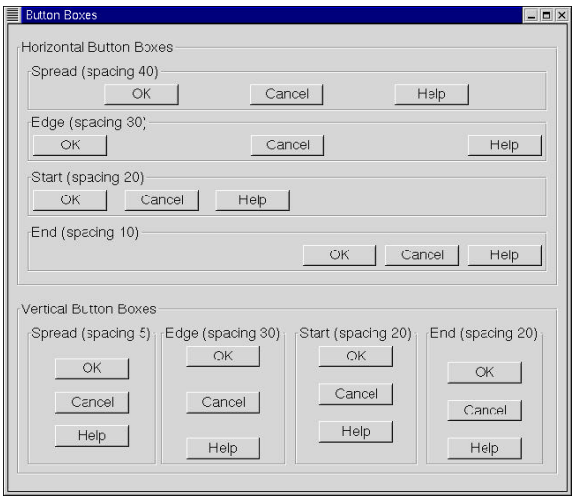

图10-6 按钮盒

10.9 工具条构件GtkToolbar

GtkToolbar(工具条构件)是非常常见的构件。它实际上是一个容器，你在上面可以添加各种各样的其他构件。最常见的情况是在上面添加普通按钮构件、开关按钮构件、无线按钮构件等。当然，也可以添加GtkTree等构件。不过，这样可能会使工具条看起来有点古怪。

应用程序的图形用户接口一般都会向同一个功能或者操作提供多种访问方式。例如，打开文件的功能可以通过“文件/打开”菜单项完成，或者通过快捷键Ctrl＋O（同时按下Ctrl键和字母O键），或者通过点击工具条上的“打开”按钮。工具条将一些构件分组，这样能够简化定制它们的外观和布局。典型情况下工具条由带图标和标签以及工具提示的按钮组成，不过，其他构件也可以放在工具条里面。按钮可以水平或垂直排列，还可以显示图标或标签，或者两者都显示。

一般情况下，在普通GtkWindow构件上创建的工具条是不能浮动的，也就是说，它不能脱离窗口。如果使用GnomeApp构件作为应用程序的主窗口，在GnomeApp上就以一个GtkHandleBox作为容器容纳工具条，这样创建的工具条可以拖动离开主窗口，浮动在桌面上，可以给人非常新颖的感觉。另外，用下面的方法创建工具条比较复杂。Gnome库提供了一种GnomeUIInfo模板，用于创建工具条和菜单非常方便。请参看后面的相关章节。

用以下函数创建工具条：

```
GtkWidget *gtk_toolbar_new( GtkOrientation orientation,
                            GtkToolbarStyle  style );
```

Orientation参数可以取以下值：

GTK_ORIENTATION_HORIZONTAL

GTK_ORIENTATION_VERTICAL

Style参数可以取以下值：

GTK_TOOLBAR_TEXT

GTK_TOOLBAR_ICONS

GTK_TOOLBAR_BOTH

style参数适用于所有用与“item”相关的函数创建的按钮(不包括作为分离构件插入到工具条的按钮)。

创建工具条以后，可以向其中追加、前插和插入工具条项（这里意指简单文本字符串或元素（这里意指任何构件类型）。要想描述一个工具条上的对象，需要一个标签文本、一个工具提示文本、一个私有工具提示文本、一个图标和一个回调函数。例如，要前插或追加一个按钮，应该使用下面的函数。

向工具条上追加（添加在其他所有构件的后面）构件：

```
GtkWidget *gtk_toolbar_append_item( GtkToolbar     *toolbar,
                                    const char     *text,
                                    const char     *tooltip_text,
                                    const char     *tooltip_private_text,
                                    GtkWidget      *icon,
                                    GtkSignalFunc  callback,
                                    gpointer       user_data );
```

向工具条上前插（添加在其他所有构件的前面）构件：

```
GtkWidget *gtk_toolbar_prepend_item( GtkToolbar    *toolbar,
                                     const char    *text,
                                     const char    *tooltip_text,
                                     const char    *tooltip_private_text,
                                     GtkWidget     *icon,
                                     GtkSignalFunc  callback,
                                     gpointer       user_data );
```

如果要使用gtk_toolbar_insert_item函数，除上面函数中要指定的参数以外，还要指定插入对象的位置，形式如下：

```
GtkWidget *gtk_toolbar_insert_item( GtkToolbar    *toolbar,
                                    const char    *text,
                                    const char    *tooltip_text,
                                    const char    *tooltip_private_text,
                                    GtkWidget     *icon,
                                    GtkSignalFunc  callback,
                                    gpointer       user_data,
                                    gint           position );
```

要简化在工具条项之间添加空白区域，可以使用下面的函数。

在工具条上追加空白区域：

```
void gtk_toolbar_append_space( GtkToolbar *toolbar );
```

在工具条上前插空白区域：

```
void gtk_toolbar_prepend_space( GtkToolbar *toolbar );
```

在工具条上的position位置插入空白区域：

```
void gtk_toolbar_insert_space( GtkToolbar *toolbar,
                               gint        position );
```

可以用下面的函数设置整个工具条上的空白区域的大小：

```
void gtk_toolbar_set_space_size( GtkToolbar *toolbar,
                                 gint        space_size) ;
```

如果需要，工具条的放置方向和它的式样可以在运行时用下面的函数设置。

设置工具条的放置方向：

```
void gtk_toolbar_set_orientation( GtkToolbar     *toolbar,
                                  GtkOrientation  orientation );
```

设置工具条的样式：

```
void gtk_toolbar_set_style( GtkToolbar      *toolbar,
                            GtkToolbarStyle  style );
```

启用或者禁用工具条的工具提示文本：

```
void gtk_toolbar_set_tooltips( GtkToolbar *toolbar,
                               gint        enable );
```

其中，enable设为TRUE则启用工具提示文本，否则，禁止显式工具提示文本。

上面第一个函数中的orientation参数是枚举值，可以取GTK_ORIENTATION_HORIZONTAL或GTK_ORIENTATION_VERTICAL。style参数用于设置工具条项的外观，可以取GTK_TOOLBAR_ICONS（只显示图标），GTK_TOOLBAR_TEXT（只显示文本）或GTK_TOOLBAR_BOTH（同时显示图标和文本）。

要想了解工具条能做什么，看一看下面的程序 (在代码之间我们插入了一些解释):

```
#include <gtk/gtk.h>
#include "gtk.xpm"
/* 这个函数连接到"close"按钮或者从窗口管理器关闭窗口的事件上 */
void delete_event (GtkWidget *widget, GdkEvent *event, gpointer data)
{
  gtk_main_quit ();
}
```

上面的代码和其他的GTK应用程序差别不大，有一点不同的是：我们包含了一个漂亮的XPM图片，用作所有按钮的图标。

```
GtkWidget* close_button; 按钮，引发一个信号以关闭应用程序 */
GtkWidget* tooltips_button; 启用/禁用工具提示 */
GtkWidget* text_button,
         * icon_button,
         * both_button; /无线按钮 */
GtkWidget* entry; /*个文本输入构件，用于显示组装到工具条上的构件 */
```

事实上，不是上面所有的构件都是必须的，我把它们放在一起，是为了让事情更清晰。

```
/* 当按钮进行状态切换时，我们检查哪一个按钮是活动的，依此设置工具条的式样
 * 注意，工具条是作为用户数据传递到回调函数的 */
void radio_event (GtkWidget *widget, gpointer data)
{
  if (GTK_TOGGLE_BUTTON (text_button)->active)
    gtk_toolbar_set_style(GTK_TOOLBAR ( data ), GTK_TOOLBAR_TEXT);
  else if (GTK_TOGGLE_BUTTON (icon_button)->active)
    gtk_toolbar_set_style(GTK_TOOLBAR ( data ), GTK_TOOLBAR_ICONS);
  else if (GTK_TOGGLE_BUTTON (both_button)->active)
    gtk_toolbar_set_style(GTK_TOOLBAR ( data ), GTK_TOOLBAR_BOTH);
}

/* 检查给定开关按钮的状态，依此启用或禁用工具提示 */
void toggle_event (GtkWidget *widget, gpointer data)
{
  gtk_toolbar_set_tooltips (GTK_TOOLBAR ( data ),
                            GTK_TOGGLE_BUTTON (widget)->active );
}
```

上面是当工具条上的一个按钮被按下时要调用的两个回调函数。

```
int main (int argc, char *argv[])
{
  /* 下面创建主窗口（一个对话框）和一个GtkHandle构件*/
  GtkWidget* dialog;
  GtkWidget* handlebox;
  GtkWidget * toolbar;
  GdkPixmap * icon;
  GdkBitmap * mask;
  GtkWidget * iconw;

  /* 初始化GTK */
  gtk_init (&argc, &argv);

  /* 用给定的标题和尺寸创建一个新窗口 */
```

```
dialog = gtk_dialog_new ();
gtk_window_set_title ( GTK_WINDOW ( dialog ) , "GTKToolbar Tutorial");
gtk_widget_set_usize( GTK_WIDGET ( dialog ) , 600 , 300 );
GTK_WINDOW ( dialog ) ->allow_shrink = TRUE;

/* 在关闭窗口时退出*/
gtk_signal_connect ( GTK_OBJECT ( dialog ), "delete_event",
                     GTK_SIGNAL_FUNC ( delete_event ), NULL);

/* 需要显现窗口，因为我们要在它的环境中为工具条设置图片 */
gtk_widget_realize ( dialog );

/* 我们将工具条放在一个手柄构件（GtkHandle）上,
 * 这样它可以从主窗口上移开*/
handlebox = gtk_handle_box_new ();
gtk_box_pack_start ( GTK_BOX ( GTK_DIALOG(dialog)->vbox ),
                     handlebox, FALSE, FALSE, 5 );
```

上面的代码和任何其他Gtk应用程序都差不多。它们进行GTK初始化，创建主窗口等。唯一需要解释的是：GtkHandlebox(手柄盒构件)。手柄盒构件只是一个可以在其中组装构件的盒子。它和普通盒子的区别在于它能从一个父窗口移开（事实上，手柄盒构件保留在父窗口上，但是它缩小为一个非常小的矩形，同时它的所有内容重新放在一个新的可自由移动的浮动窗口上)。拥有一个可浮动工具条给人感觉非常好，所以这两种构件经常同时使用。

```
/* 工具条设置为水平的，同时带有图标和文本
 * 在每个项之间有5像素的间距,
 * 并且，我们将它们全部放在手柄盒构件上 */
toolbar = gtk_toolbar_new ( GTK_ORIENTATION_HORIZONTAL,
                            GTK_TOOLBAR_BOTH );
gtk_container_set_border_width ( GTK_CONTAINER ( toolbar ) , 5 );
gtk_toolbar_set_space_size ( GTK_TOOLBAR ( toolbar ), 5 );
gtk_container_add ( GTK_CONTAINER ( handlebox ) , toolbar );

icon = gdk_pixmap_create_from_xpm_d ( dialog->window, &mask,
    &dialog->style->white, gtk_xpm );
```

上面的代码初始化工具条，并创建了一个GDKPixmap图片。

```
/* 工具条上第一项是"close"按钮*/
  iconw = gtk_pixmap_new ( icon, mask )图标构件*/
  close_button =
  gtk_toolbar_append_item ( GTK_TOOLBAR (toolbar工具条*/
                    "Close",                按钮标签*/
                    "Closes this app",      按钮的工具提示*/
                    "Private",              工具提示私有信息*/
                     iconw,                  图标构件 */
                     GTK_SIGNAL_FUNC (delete_event),一个信号 */
                      NULL );
  gtk_toolbar_append_space ( GTK_TOOLBAR ( toolbar )按钮之间的空白*/
```

在上面的代码中，可以看到最简单的情况：在工具条上增加一个按钮。在追加一个新的工具条项前，必须构造一个pixmap构件用作该项的图标，这个步骤我们要对每一个工具条项

重复一次。在工具条项之间还要增加分隔空间，这样后面的工具条项就不会一个接一个紧挨着。可以看到，gtk_toolbar_append_item返回一个指向新创建的按钮构件的指针，所以我们可以用正常的方式使用它。

```
/* 现在，我们创建无线按钮组*/
iconw = gtk_pixmap_new ( icon, mask );
icon_button = gtk_toolbar_append_element(
            GTK_TOOLBAR(toolbar),
            GTK_TOOLBAR_CHILD_RADIOBUTTON,元素的类型*/
            NULL,                          指向构件的指针 */
            "Icon",                        标签 */
            "Only icons in toolbar",       工具提示*/
            "Private",                     工具提示私有字符串*/
            iconw,                         图标 */
            GTK_SIGNAL_FUNC (radio_event),信号 */
            toolbar);                      信号传递的数据*/
gtk_toolbar_append_space ( GTK_TOOLBAR ( toolbar ) );
```

这里我们通过使用gtk_toolbar_append_element函数创建了一个无线按钮组。事实上，使用这个函数，我们能够添加简单的工具条项，或者设置添加按钮间的分隔条 (type = GTK_TOOLBAR_CHILD_SPACE 或GTK_TOOLBAR_CHILD_BUTTON)。在上面的例子中，我们先创建了一个无线按钮组。要为这个组创建其他按钮，需要一个指向前一个按钮的指针，以便可以很容易地创建一列按钮。

```
/* 下面引用的无线按钮就是前面创建的按钮 */
iconw = gtk_pixmap_new ( icon, mask );
text_button =
  gtk_toolbar_append_element(GTK_TOOLBAR(toolbar),
                             GTK_TOOLBAR_CHILD_RADIOBUTTON,
                             icon_button,
                             "Text",
                             "Only texts in toolbar",
                             "Private",
                             iconw,
                             GTK_SIGNAL_FUNC (radio_event),
                             toolbar);
gtk_toolbar_append_space ( GTK_TOOLBAR ( toolbar ) );

iconw = gtk_pixmap_new ( icon, mask );
both_button =
  gtk_toolbar_append_element(GTK_TOOLBAR(toolbar),
                             GTK_TOOLBAR_CHILD_RADIOBUTTON,
                             text_button,
                             "Both",
                             "Icons and text in toolbar",
                             "Private",
                             iconw,
                             GTK_SIGNAL_FUNC (radio_event),
                             toolbar);
gtk_toolbar_append_space ( GTK_TOOLBAR ( toolbar ) );
gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(both_button),TRUE);
```

最后，我们必须手工设置其中一个按钮的状态 (否则它们全部保留为活动状态，并阻止我们在它们之间进行状态切换)。

```
    /* 下面只是一个简单的开关按钮 */
    iconw = gtk_pixmap_new ( icon, mask );
    tooltips_button =
      gtk_toolbar_append_element(GTK_TOOLBAR(toolbar),
                                 GTK_TOOLBAR_CHILD_TOGGLEBUTTON,
                                 NULL,
                                 "Tooltips",
                                 "Toolbar with or without tips",
                                 "Private",
                                 iconw,
                                 GTK_SIGNAL_FUNC (toggle_event),
                                 toolbar);
    gtk_toolbar_append_space ( GTK_TOOLBAR ( toolbar ) );
    gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(tooltips_button),TRUE);

    /* 要将一个构件组装到工具条上，只需创建它，然后将它追
     * 加到工具条上，同时设置合适的工具提示 */
    entry = gtk_entry_new ();
    gtk_toolbar_append_widget( GTK_TOOLBAR (toolbar),
                               entry,
                               "This is just an entry",
                               "Private" );

    /* 因为它不是在工具条上创建的，所以我们必须显示它 */
    gtk_widget_show ( entry );
```

可以看到，将任何构件添加到工具条上都是非常简单的。唯一要记住的是，这个构件必须手工显示(与此相反，在工具条内创建的工具条项随工具条一起显示)。

```
    /* 好了，现在可以显示所有的东西了 */
    gtk_widget_show ( toolbar );
    gtk_widget_show (handlebox);
    gtk_widget_show ( dialog );

    /* 进入主循环，等待用户的操作 */
    gtk_main ();

    return 0;
}
```

上面就是工具条教程的全部代码。当然，还需要一个漂亮的 XPM图片。下面就是一个：

```
/* XPM */
static char * gtk_xpm[] = {
"32 39 5 1",
".      c none",
"+      c black",
"@      c #3070E0",
"#      c #F05050",
"$      c #35E035",
```

```
".................+.............",
".............+++++.............",
"............+++++@@++...........",
".........+++++@@@@@@++.........",
"........++++@@@@@@@@@@++........",
"......++++@@++++++++@@@++.......",
".....+++@@@++++++++++@@@++.....",
"...+++@@@@+++@@@@@@++++@@@@+....",
"..+++@@@@+++@@@@@@@@+++@@@@@++..",
".++@@@@@@+++@@@@@@@@@@@@@@@@@@++",
".+#+@@@@@@++@@@@+++@@@@@@@@@@@@@+",
".+###++@@@@+++@@@+++++@@@@@@@@$@.",
".+#####+@@@@+++@@@+++@@@@@++$$$@.",
".+########+@@@+++++++@@@@@+@$$$$@.",
".+#########+@@@@+++@@@@++@$$$$$$+.",
".+##########++@@@@@@@++@$$$$$$$$+.",
".+########+###+@@@@+++$$$$$$@@$$+.",
".+########+###+@@++@$$$$$$++$$$+.",
".+#########+###+@@$$$$$$$$@+@$$@+.",
".+##########++++@$$@+@$$@++$$$@+.",
".+##############++$$@+@$$++$$$$+..",
".++#############++$$@+@$++@$$$$+..",
".+#############+###++$$++@+++$$$$$+..",
".++#########+##+#++$$+++++@$$$$$+..",
".++########++#####++$$++++++@$$$@+..",
".+######++######++$$+++@++++@$@+..",
".+######++######++$$++@$$@+++$@@..",
".++######++######++$$+++$$$$$+@$@++.",
".++######++######++$$++$$$$$$$$$+++.",
".++#####+######++$$++$$$$$$$@+++.",
"..+++++++++++++@$$+@$$$$$$+++...",
"...+++++++++++@$$$$$$$$$@+++....",
".....+++#######+@$$$$$$$$+++......",
"......+++######+@$$$$$@++........",
".......+++####+@$$$$+++.........",
".........++###+$$$@++...........",
"..........++##+$@+++............",
"...........+++++++..............",
".............++++...............";
```

10.10 笔记本构件GtkNotebook

GtkNotebook(笔记本构件)是互相重叠的页面集合，每一页都包含不同的信息，且一次只有一个页面是可见的。该构件在 GUI(图形用户接口)编程中很常用。要说明大量的相似信息，同时把它们分别显示时，使用这种构件是一个很好的方法。许多应用程序的“选项”对话框都使用了这个构件。

用下面的函数可以创建新笔记本构件。

```
GtkWidget *gtk_notebook_new( void );
```

一旦创建了笔记本构件，就可以使用一系列的函数操作该构件。下面将对它们进行分别

讨论。

先看一下怎样定位页面指示器——或称页标签，可以有四种位置：

```
void gtk_notebook_set_tab_pos( GtkNotebook     *notebook,
                               GtkPositionType  pos );
```

GtkPositionType参数可以取以下几个值，从字面上很容易理解它们的含义：

GTK_POS_LEFT　　将标签页放在左边
GTK_POS_RIGHT　　将标签页放在右边
GTK_POS_TOP　　将标签页放在顶部
GTK_POS_BOTTOM　　将标签页放在底部

它的缺省值是GTK_POS_TOP。

下面看一下怎样向笔记本中添加页面。有三种方法向笔记本中添加页面。前两种方法是非常相似的。

在笔记本构件中追加页面：

```
void gtk_notebook_append_page( GtkNotebook *notebook,
                               GtkWidget   *child,
                               GtkWidget   *tab_label );
```

在笔记本中前插页面：

```
void gtk_notebook_prepend_page( GtkNotebook *notebook,
                                GtkWidget   *child,
                                GtkWidget   *tab_label );
```

child 参数是放在笔记本上的子构件， tab_label是要添加的页面的标签。子构件必须分别创建，一般是一个容器构件，比如说表格构件。

最后一个函数与前两个函数类似，不过允许指定页面加入的位置。

```
void gtk_notebook_insert_page( GtkNotebook *notebook,
                               GtkWidget   *child,
                               GtkWidget   *tab_label,
                               gint         position );
```

其中的参数与_append_ and _prepend_函数一样，还包含一个额外参数——插入位置。该参数指定页面应该插入到哪一页。注意，第一页位置为 0。

前面介绍了怎样添加页面，下面介绍怎样删除页面。

```
void gtk_notebook_remove_page( GtkNotebook *notebook,
                               gint         page_num );
```

这个函数从笔记本中删除由page_num参数指定的页面。

用以下函数寻找笔记本构件的当前标签页：

```
gint gtk_notebook_get_current_page( GtkNotebook *notebook );
```

下面两个函数将笔记本构件的页面向前或向后移动。对要操作的笔记本构件使用以下函数就可以了。

注意　当笔记本构件在最后一页时，调用gtk_notebook_next_page 函数，笔记本构件会跳到第一页。同样，如果笔记本构件在第一页，调用了函数 gtk_notebook_prev_page，笔记本构件会跳到最后一页。

```
void gtk_notebook_next_page( GtkNoteBook *notebook );
```

```
void gtk_notebook_prev_page( GtkNoteBook *notebook );
```

下面的函数设置“活动”页面。缺省状态下，笔记本显示第一页。

```
void gtk_notebook_set_page( GtkNotebook *notebook,
                            gint         page_num );
```

下面的函数显示或隐藏笔记本构件的页标签以及它的边框。

```
void gtk_notebook_set_show_tabs( GtkNotebook *notebook,
                                 gboolean     show_tabs);
void gtk_notebook_set_show_border( GtkNotebook *notebook,
                                   gboolean     show_border );
```

第一个函数中的show_tabs为TRUE则显示页标签，FALSE不显示页标签。

第二个函数中的show_border为TRUE则显示边框，为FALSE不显示边框。

如果页面较多，标签页在页面上排列不下时，可以用下面的函数。它允许标签页用两个按钮箭头滚动。

```
void gtk_notebook_set_scrollable( GtkNotebook *notebook,
                                  gboolean     scrollable );
```

show_tabs, show_border 和scrollable参数可以是TRUE或FALSE。

下面是一个关于GtkBooknote构件的示例。这个程序创建了一个带一个笔记本构件和 6个按钮的窗口。笔记本构件包含 11页，你可以用三种方式对它们进行添加：追加、前插、插入。点击按钮可以改变标签的位置，添加、删除标签和边框，删除一页，向前或向后改变标签页，以及退出程序。

```
/* 笔记本构件示例开始notebook.c */

#include <gtk/gtk.h>

/* 这个函数旋转标签页的位置 */
void rotate_book (GtkButton *button, GtkNotebook *notebook)
{
    gtk_notebook_set_tab_pos (notebook, (notebook->tab_pos +1) %4);
}

/* 添加/删除页标签和边框*/
void tabsborder_book (GtkButton *button, GtkNotebook *notebook)
{
    gint tval = FALSE;
    gint bval = FALSE;
    if (notebook->show_tabs == 0)
            tval = TRUE;
    if (notebook->show_border == 0)
            bval = TRUE;

    gtk_notebook_set_show_tabs (notebook, tval);
    gtk_notebook_set_show_border (notebook, bval);
}

/* 从笔记本构件上删除页面 */
void remove_book (GtkButton *button, GtkNotebook *notebook)
{
```

```
    gint page;
    page = gtk_notebook_get_current_page(notebook);
    gtk_notebook_remove_page (notebook, page);
    /* 必须刷新构件 ——
     这会迫使构件重绘自身*/
    gtk_widget_draw(GTK_WIDGET(notebook), NULL);
}

void delete (GtkWidget *widget, GtkWidget *event, gpointer data)
{
    gtk_main_quit ();
}

int main (int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *button;
    GtkWidget *table;
    GtkWidget *notebook;
    GtkWidget *frame;
    GtkWidget *label;
    GtkWidget *checkbutton;
    int i;
    char bufferf[32];
    char bufferl[32];

    gtk_init (&argc, &argv);
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (delete), NULL);

    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    table = gtk_table_new(3,6,FALSE);
    gtk_container_add (GTK_CONTAINER (window), table);

    /* 创建一个新的笔记本构件，将标签页放在顶部 */
    notebook = gtk_notebook_new ();
    gtk_notebook_set_tab_pos (GTK_NOTEBOOK (notebook), GTK_POS_TOP);
    gtk_table_attach_defaults(GTK_TABLE(table), notebook, 0,6,0,1);
    gtk_widget_show(notebook);

    /* 在笔记本构件后面追加几个页面*/
    for (i=0; i < 5; i++) {
        sprintf(bufferf, "Append Frame %d", i+1);
        sprintf(bufferl, "Page %d", i+1);

        frame = gtk_frame_new (bufferf);
        gtk_container_set_border_width (GTK_CONTAINER (frame), 10);
        gtk_widget_set_usize (frame, 100, 75);
        gtk_widget_show (frame);
```

```
    label = gtk_label_new (bufferf);
    gtk_container_add (GTK_CONTAINER (frame), label);
    gtk_widget_show (label);

    label = gtk_label_new (bufferl);
    gtk_notebook_append_page (GTK_NOTEBOOK (notebook), frame, label);
}

/* 在指定位置添加页面 */
checkbutton = gtk_check_button_new_with_label ("Check me please!");
gtk_widget_set_usize(checkbutton, 100, 75);
gtk_widget_show (checkbutton);

label = gtk_label_new ("Add page");
gtk_notebook_insert_page (GTK_NOTEBOOK (notebook), checkbutton, label, 2);

/* 向笔记本构件前插标签页 */
for (i=0; i < 5; i++) {
    sprintf(bufferf, "Prepend Frame %d", i+1);
    sprintf(bufferl, "PPage %d", i+1);

    frame = gtk_frame_new (bufferf);
    gtk_container_set_border_width (GTK_CONTAINER (frame), 10);
    gtk_widget_set_usize (frame, 100, 75);
    gtk_widget_show (frame);

    label = gtk_label_new (bufferf);
    gtk_container_add (GTK_CONTAINER (frame), label);
    gtk_widget_show (label);

    label = gtk_label_new (bufferl);
    gtk_notebook_prepend_page (GTK_NOTEBOOK(notebook), frame, label);
}

/* 设置起始页(第4页) */
gtk_notebook_set_page (GTK_NOTEBOOK(notebook), 3);

/* 创建一排按钮 */
button = gtk_button_new_with_label ("close");
gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
                           GTK_SIGNAL_FUNC (delete), NULL);
gtk_table_attach_defaults(GTK_TABLE(table), button, 0,1,1,2);
gtk_widget_show(button);

button = gtk_button_new_with_label ("next page");
gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
                           (GtkSignalFunc) gtk_notebook_next_page,
                           GTK_OBJECT (notebook));
gtk_table_attach_defaults(GTK_TABLE(table), button, 1,2,1,2);
gtk_widget_show(button);
```

```
    button = gtk_button_new_with_label ("prev page");
    gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
                           (GtkSignalFunc) gtk_notebook_prev_page,
                           GTK_OBJECT (notebook));
    gtk_table_attach_defaults(GTK_TABLE(table), button, 2,3,1,2);
    gtk_widget_show(button);

    button = gtk_button_new_with_label ("tab position");
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                        (GtkSignalFunc) rotate_book,
                        GTK_OBJECT(notebook));
    gtk_table_attach_defaults(GTK_TABLE(table), button, 3,4,1,2);
    gtk_widget_show(button);

    button = gtk_button_new_with_label ("tabs/border on/off");
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                        (GtkSignalFunc) tabsborder_book,
                        GTK_OBJECT (notebook));
    gtk_table_attach_defaults(GTK_TABLE(table), button, 4,5,1,2);
    gtk_widget_show(button);

    button = gtk_button_new_with_label ("remove page");
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                        (GtkSignalFunc) remove_book,
                        GTK_OBJECT(notebook));
    gtk_table_attach_defaults(GTK_TABLE(table), button, 5,6,1,2);
    gtk_widget_show(button);

    gtk_widget_show(table);
    gtk_widget_show(window);
    gtk_main ();
    return(0);
}
/* 示例结束 */
```

运行效果见图 10-7。请尝试一下添加、删除页面，切换页标签的位置，以及显示、隐藏页标签等功能。

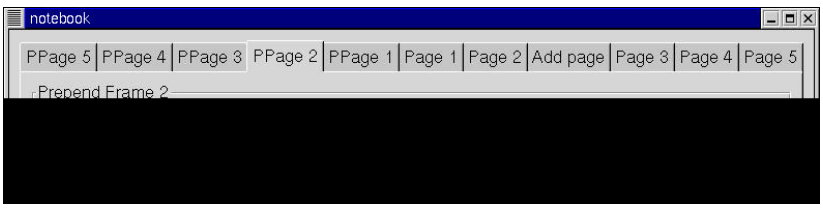

图10-7 笔记本构件